

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void inversematrix(float matrix[d][d], float inverse[d][d]) {
    for (int i = 0; i < d; i++) {
        for (int j = 0; j < d; j++) {
            if (i == j) inverse[i][j] = 1.0;
            else inverse[i][j] = 0.0;
        }
    }
    for (int i = 0; i < d; i++) {
        int pivot = i;
        for (int j = i + 1; j < d; j++) {
            if (fabs(matrix[j][i]) > fabs(matrix[pivot][i])) {
                pivot = j;
            }
        }
    }

    float temp[d];
    for (int k = 0; k < d; k++) {
        float t = matrix[i][k];
        matrix[i][k] = matrix[pivot][k];
        matrix[pivot][k] = t;

        t = inverse[i][k];
        inverse[i][k] = inverse[pivot][k];
        inverse[pivot][k] = t;
    }

    if (fabs(matrix[i][i]) < 1e-9) {
        printf("The matrix is singular\n");
        exit(1);
    }

    float divisor = matrix[i][i];
    for (int j = 0; j < d; j++) {
        matrix[i][j] /= divisor;
        inverse[i][j] /= divisor;
    }

    for (int k = 0; k < d; k++) {
        if (k != i) {
            float factor = matrix[k][i];
```

```

        for (int j = 0; j < d; j++) {
            matrix[k][j] -= factor * matrix[i][j];
            inverse[k][j] -= factor * inverse[i][j];
        }
    }
}
}

int main(){

    // Column 0 is always 1
    float X[n][d];

    float y[n][1];

    float X_transpose[d][n];
    for(int i=0; i<n; i++){
        for(int j=0; j<d; j++){
            X_transpose[j][i] = X[i][j];
        }
    }

    float X_transpose_X_product[d][d];
    float sum = 0;

    for(int i=0; i<d; i++){
        for(int j=0; j<d; j++){
            sum = 0;
            for(int k=0; k<n; k++){
                sum += X_transpose[i][k] * X[k][j];
            }
            X_transpose_X_product[i][j] = sum;
        }
    }

    float X_transpose_y_product[d][1];

    for(int i=0; i<d; i++){
        for(int j=0; j<1; j++){
            sum = 0;
            for(int k=0; k<n; k++){
                sum += X_transpose[i][k] * y[k][j];
            }
            X_transpose_y_product[i][j] = sum;
        }
    }
}

```

```

}

float X_transpose_X_product_inverse[d][d];
inversematrix(X_transpose_X_product, X_transpose_X_product_inverse);

float final_weights[d][1];

for(int i=0; i<d; i++){
    for(int j=0; j<1; j++){
        sum = 0;
        for(int k=0; k<d; k++){
            sum += X_transpose_X_product_inverse[i][k] * X_transpose_y_product[k][j];
        }
        final_weights[i][j] = sum;
    }
}

int choice;
float input_val;
float prediction;

while(1) {
    printf("1. Predict\n");
    printf("2. Exit\n");
    printf("Enter choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("Enter value: ");
        scanf("%f", &input_val);

        prediction = final_weights[0][0] + (final_weights[1][0] * input_val);

        printf("Predicted value: %.3f\n", prediction);
    }
    else if (choice == 2) {
        printf("ngl gang ts pmo\n");
        break;
    }
    else {
        printf("Invalid\n");
    }
}
}

```

Derivation of the Normal Equation with Dimensions

Definitions:

- n : Number of samples (rows)
- d : Number of features (columns)
- $X_{(n \times d)}$: Input Matrix
- $\mathbf{y}_{(n \times 1)}$: Target Vector
- $\mathbf{w}_{(d \times 1)}$: Weights Vector

1. The Cost Function

We minimize the Squared Error (Residual Sum of Squares):

$$J(\mathbf{w}) = \|X_{(n \times d)}\mathbf{w}_{(d \times 1)} - \mathbf{y}_{(n \times 1)}\|^2$$

$$J(\mathbf{w}) = (X_{(n \times d)}\mathbf{w}_{(d \times 1)} - \mathbf{y}_{(n \times 1)})^T (X_{(n \times d)}\mathbf{w}_{(d \times 1)} - \mathbf{y}_{(n \times 1)})$$

2. Expansion

Apply the transpose $(A - B)^T = A^T - B^T$:

$$J(\mathbf{w}) = (\mathbf{w}_{(1 \times d)}^T X_{(d \times n)}^T - \mathbf{y}_{(1 \times n)}^T) (X_{(n \times d)}\mathbf{w}_{(d \times 1)} - \mathbf{y}_{(n \times 1)})$$

3. Multiplication (FOIL)

Expand the terms. Note that scalar terms are equal to their own transpose.

$$J(\mathbf{w}) = \mathbf{w}_{(1 \times d)}^T X_{(d \times n)}^T X_{(n \times d)}\mathbf{w}_{(d \times 1)} - \underbrace{\mathbf{w}_{(1 \times d)}^T X_{(d \times n)}^T \mathbf{y}_{(n \times 1)}}_{\text{Scalar } (1 \times 1)}$$

$$- \underbrace{\mathbf{y}_{(1 \times n)}^T X_{(n \times d)}\mathbf{w}_{(d \times 1)}}_{\text{Scalar } (1 \times 1)} + \mathbf{y}_{(1 \times n)}^T \mathbf{y}_{(n \times 1)}$$

Combine the middle terms:

$$J(\mathbf{w}) = \mathbf{w}_{(1 \times d)}^T (X^T X)_{(d \times d)} \mathbf{w}_{(d \times 1)} - 2\mathbf{w}_{(1 \times d)}^T (X^T \mathbf{y})_{(d \times 1)} + \mathbf{y}^T \mathbf{y}$$

4. Derivative w.r.t \mathbf{w}

Set the gradient to zero to find the minimum:

$$\frac{\partial J}{\partial \mathbf{w}} = 2(X^T X)_{(d \times d)} \mathbf{w}_{(d \times 1)} - 2(X^T \mathbf{y})_{(d \times 1)} = \mathbf{0}_{(d \times 1)}$$

5. Final Solution

Isolate \mathbf{w} :

$$(X^T X)_{(d \times d)} \mathbf{w}_{(d \times 1)} = (X^T \mathbf{y})_{(d \times 1)}$$
$$\mathbf{w}_{(d \times 1)} = ((X^T X)_{(d \times d)})^{-1} (X^T \mathbf{y})_{(d \times 1)}$$

The Setup: Turning Data into Matrices

We assume a dataset of 3 houses where the price (y) is exactly twice the size (x).

- House 1: Size = 1, Price = 2
- House 2: Size = 2, Price = 4
- House 3: Size = 3, Price = 6

Defining the Matrices

We add a column of **1s** to the input matrix X to account for the bias term (w_0).

$$X_{(3 \times 2)} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad \mathbf{y}_{(3 \times 1)} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$

(Columns of X : Bias, Size)

Step-by-Step Derivation

We use the Normal Equation to find the weight vector \mathbf{w} :

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$

Step 1: Calculate $X^T X$ (Feature Matrix)

First, we compute the inner product of the input features. This captures the variance and scale of the data.

$$X^T X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} (1+1+1) & (1+2+3) \\ (1+2+3) & (1+4+9) \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

Step 2: Calculate $X^T \mathbf{y}$ (Correlation Vector)

Next, we map the relationship between the features and the target prices.

$$X^T \mathbf{y} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} (2+4+6) \\ (2+8+18) \end{bmatrix} = \begin{bmatrix} 12 \\ 28 \end{bmatrix}$$

Step 3: Calculate the Inverse $(X^T X)^{-1}$

To isolate \mathbf{w} , we must invert the matrix from Step 1.
The determinant is: $(3 \times 14) - (6 \times 6) = 42 - 36 = 6$.

$$(X^T X)^{-1} = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix}$$

Solving for Weights (\mathbf{w})

Finally, we multiply the Inverse by the Correlation Vector:

$$\mathbf{w} = \text{Inverse} \times (X^T \mathbf{y})$$

$$\mathbf{w} = \frac{1}{6} \left(\begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} 12 \\ 28 \end{bmatrix} \right)$$

Calculating the rows:

$$\begin{aligned} \text{Row 1: } (14 \times 12) + (-6 \times 28) &= 168 - 168 = 0 \\ \text{Row 2: } (-6 \times 12) + (3 \times 28) &= -72 + 84 = 12 \end{aligned}$$

$$\mathbf{w} = \frac{1}{6} \begin{bmatrix} 0 \\ 12 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

Result:

- Bias (w_0) = 0
- Slope (w_1) = 2

The learned equation is: Price = 0 + 2(Size).

Making a Prediction

We predict the price for a new house with **Size = 5**.

The input vector is $\mathbf{x}_{new} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$ (bias, size).

$$y_{pred} = \mathbf{w}^T \mathbf{x}_{new}$$

$$y_{pred} = [0 \ 2] \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

$$y_{pred} = (0 \times 1) + (2 \times 5) = 10$$

The model correctly predicts the price is **10**.